Лабораторная работа №4:
Поиск контуров и примитивов на изображении

Цель:

Целью данной работы является изучение алгоритмов поиска контуров и примитивов на изображениях.

Задание:

Необходимо разработать приложение Windows Forms, способное осуществлять:

1. Обнаружение контуров.
2. Обнаружение примитивов (окружность, треугольник, четырёхугольник).

Поиск контуров:

Первым этапом поиска контуров является **устранение шумов** на изображении. Как правило, это осуществляется при помощи преобразования изображения в чёрно-белое и применение к нему размытия.

В 1 и 2 лабораторных работах были рассмотрены несколько способов устранения шумов - пирамидальная декомпозиция (gaussian pyramid decomposition) и медианный фильтр (median blur). Ещё один популярный способ – **размытие** по Гауссу (gaussian blur) – реализуется следующим образом:

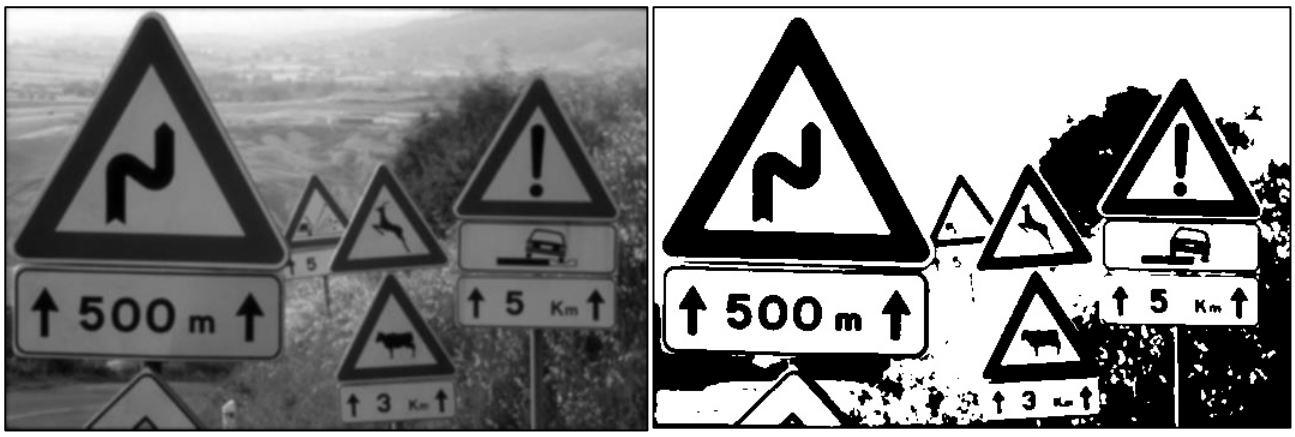
```
var grayImage = sourceImage.Convert<Gray, byte>();
int kernelSize = 5; // радиус размытия
var bluredImage = grayImage.SmoothGaussian(kernelSize);
```

Результат преобразования:

Следующий этап – **обнаружение областей интереса**. Простейшим способом обнаружения таких областей является пороговое преобразование изображения (бинаризация):

```
var threshold = new Gray(80); // пороговое значение
var color = new Gray(255); // этим цветом будут закрашены пиксели, имеющие значение > threshold
var binarizedImage = bluredImage.ThresholdBinary(threshold, color);
```

Результат преобразования:



Последним этапом является **нахождение контуров**. Контур описывает некоторую границу на изображении в виде вектора точек. Нахождение контуров осуществляется следующим образом:

```
var contours = new VectorOfVectorOfPoint(); // контейнер для хранения контуров
CvInvoke.FindContours(
       binarizedImage, // исходное чёрно-белое изображение
       contours, // найденные контуры
       null, // объект для хранения иерархии контуров (в данном случае не используется)
       RetrType.List, // структура возвращаемых данных (в данном случае список)
       ChainApproxMethod.ChainApproxSimple); // метод аппроксимации (сжимает горизонтальные,
                                             //вертикальные и диагональные сегменты
                                             //и оставляет только их конечные точки)
```

После нахождения контуров их можно **нарисовать**, используя следующий код:

```
var contoursImage = sourceImage.CopyBlank(); //создание "пустой" копии исходного изображения
for (int i = 0; i < contours.Size; i++)
{
       var points = contours[i].ToArray();
       contoursImage.Draw(points, new Bgr(Color.GreenYellow), 2); // отрисовка точек
}
```

Результат отрисовки контуров при пороговом значении `threshold = 80`:

Поиск примитивов:

Для того чтобы обнаружить геометрические примитивы среди найденных контуров, необходимо избавиться от избыточных точек, объединив их в линии. Сделать это можно при помощи функции **аппроксимации контуров**:

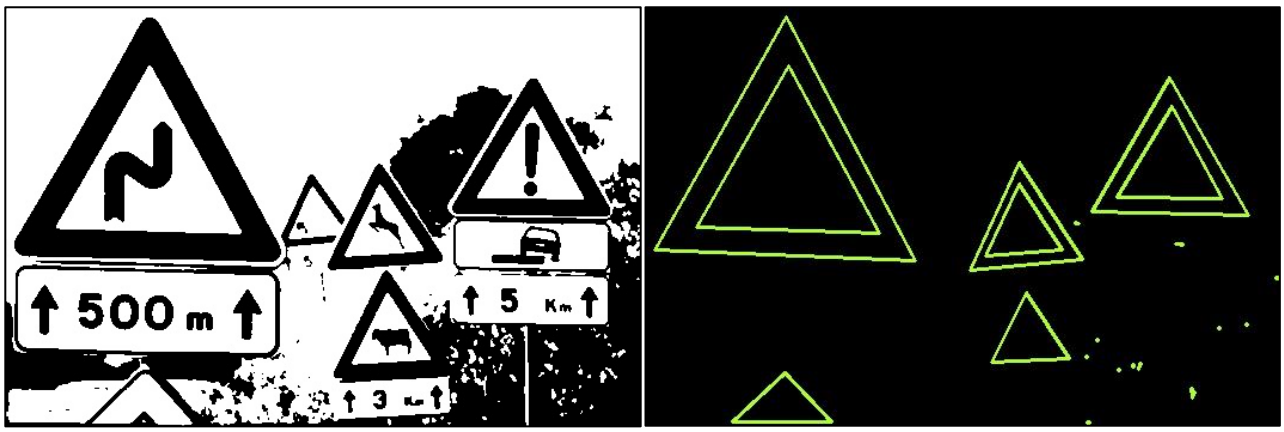
```
for (int i = 0; i < contours.Size; i++)
{
        var approxContour = new VectorOfPoint();
        CvInvoke.ApproxPolyDP(
                contours[i], // исходный контур
                approxContour, // контур после аппроксимации
                CvInvoke.ArcLength(contours[i], true) * 0.05, // точность аппроксимации, прямо
                                                              //пропорциональная площади контура
                true); // контур становится закрытым (первая и последняя точки соединяются)
}
```

Результат отрисовки аппроксимированных контуров при пороговом значении `threshold = 80`:

После выполнения аппроксимации контуров, можно обнаружить **треугольники**, основываясь на количестве точек, оставшихся в соответствующем контуре:

```
if (approxContour.Size == 3) // если контур содержит 3 точки, то рисуется треугольник
{
        var points = approxContour.ToArray();
        contoursImage.Draw(new Triangle2DF(points[0], points[1], points[2]),
                                                                new Bgr(Color.GreenYellow), 2);
}
```

Результат отрисовки контуров, состоящих из трёх точек, при пороговом значении `threshold = 80`:



От контуров, не попадающих в желаемый **диапазон площадей**, можно избавиться, используя простое условие:

```
// проверка на площадь треугольника > минимально допустимой площади
if (CvInvoke.ContourArea(approxContour, false) > minArea)
{
        ...
}
```

Результат отрисовки контуров на исходном изображении при пороговом значении `threshold = 80` и минимальной площади `minArea = 256`:

Обнаружение **четырёхугольников** осуществляется подобным образом, но с проверкой на диапазон значений углов между сторонами:

```
private bool isRectangle(Point[] points)
{
        int delta = 10; // максимальное отклонение от прямого угла
        LineSegment2D[] edges = PointCollection.PolyLine(points, true);
        for (int i = 0; i < edges.Length; i++) // обход всех ребер контура
        {
                double angle = Math.Abs(edges[(i + 1) %
                                                edges.Length].GetExteriorAngleDegree(edges[i]));
                if (angle < 90 - delta || angle > 90 + delta) // если угол непрямой
                {
                        return false;
                }
        }
        return true;
}
```

А функция отрисовки изменится на:

```
contoursImage.Draw(CvInvoke.MinAreaRect(approxContour), new Bgr(Color.GreenYellow), 2);
```

Результат работы при `threshold = 120` и `minArea = 99`:

Обнаружение окружностей является более сложной задачей, для решения которой рекомендуется использовать алгоритм Хафа (Hough). Для улучшения качества обнаружения рекомендуется привести изображение в градации серого и очистить от шумов:

```
var grayImage = sourceImage.Convert<Gray, byte>();
var bluredImage = grayImage.SmoothGaussian(9);
```

Непосредственно обнаружение окружностей можно осуществить с помощью метода `HoughCircles`. Пример:

```
List<CircleF> circles = new List<CircleF>(CvInvoke.HoughCircles(bluredImage,
            HoughModes.Gradient,
            1.0,
            minDistance,
            100,
            acTreshold,
            minRadius,
            maxRadius));
```

где:

`minDistance` – минимальная дистанция между центрами обнаруженных окружностей;

`acTreshold` – пороговое значение при определении центра окружности;

`minRadius/maxRadius` – минимальный и максимальный радиусы окружностей.

Метод `HoughCircles` возвращает **список окружностей** типа `CircleF`. Для отрисовки найденных окружностей, можно использовать следующий цикл:

```
var resultImage = sourceImage.Copy();
foreach (CircleF circle in circles) resultImage.Draw(circle, new Bgr(Color.Blue), 2);
```

Результат работы при параметрах: `minDistance = 250, acTreshold = 36`:

Поиск по цвету:

В некоторых случаях удобно выполнять поиск примитивов по цвету. Поиск по цвету заключается в выделении пикселей, принадлежащих некоторому диапазону значений в одном из каналов изображения. Для поиска синих, зеленых или красных объектов необходимо использовать B, G, и R каналы соответственно. В общем случае удобнее использовать канал Hue (цветовой тон) цветовой модели HSV.

Порядок действий описывается следующим фрагментом кода:

```
var hsvImage = sourceImage.Convert<Hsv, byte>(); // конвертация в HSV

var hueChannel = hsvImage.Split()[0]; // выделение канала Hue

byte color = 30; // соответствует желтому тону в Emgu.CV
byte rangeDelta = 10; // величина разброса цвета
var resultImage = hueChannel.InRange(new Gray(color - rangeDelta), new Gray(color + rangeDelta)); // выделение цвета
```

Метод `InRange` принимает 2 параметра – нижнюю и верхнюю границу искомых пикселей – и возвращает бинарное изображение.
Исходное изображение и результат можно увидеть ниже. Белые пиксели на результирующем изображении соответствуют искомым.

Дальнейшие преобразования осуществляются аналогично предыдущим пунктам.

Возможные улучшения:

Как можно заметить при выполнении лабораторной работы, результат работы предложенных алгоритмов сильно зависит от яркости и контраста исходного изображения, а также от параметров порогового преобразования, аппроксимации и т.д. Одним из способов повышения эффективности работы является **предварительная обработка** исходного изображения. Например, увеличение яркости и контраста изображения, повышение его резкости при помощи алгоритмов, изученных в рамках лабораторной работы номер 2.

Другим способом является использование более совершенных алгоритмов определения контуров. Например, вместо порогового преобразования может быть использован детектор границ Кэнни:

```
var cannyEdges = bluredImage.Canny(thresh, threshLinking);
```

где:

`thresh` – пороговое значение для поиска ярко выраженных границ.

`threshLinking` – пороговое значение допустимое при связывании контуров.

Сравнение результатов работы порогового преобразования (слева) и детектора границ Кэнни (справа):

Помимо этого, для разных задач и типов изображений, могут применяться разные типы порогового преобразования:

```
bluredImage.ThresholdBinaryInv(new Gray(80), new Gray(255));
binarizedImage = bluredImage.ThresholdToZero(new Gray(80));
binarizedImage = bluredImage.ThresholdToZeroInv(new Gray(80));
binarizedImage = bluredImage.ThresholdAdaptive(new Gray(255), AdaptiveThresholdType.GaussianC,
                                                ThresholdType.Binary, 3, new Gray(0.03));
binarizedImage = bluredImage.ThresholdTrunc(new Gray(80));
```

Полезным приемом при поиске примитивов является выделение **регионов интереса**. Поскольку различные области изображения имеют широкий диапазон свойств и не всегда могут быть обработаны единым набором параметров, имеет смысл проводить предварительную обработку изображения, выделять области интереса, после чего продолжать обработку каждой из этих областей с индивидуальным набором параметров.

Для нахождения ограничивающего объёма для каждого из контуров, а так же их геометрических центров, можно использовать следующий метод:

```
Rectangle rect = CvInvoke.BoundingRectangle(contours[i]);
```

Для копирования части изображения, представляющего интерес, можно использовать следующий метод:

```
image.ROI = rect; // установка региона интереса
var roiImage = image.Clone();
image.ROI = Rectangle.Empty; // сброс региона интереса
```

Задание:

Реализовать программное средство, позволяющее отображать в одном окне два изображения, «оригинальное» слева и «результат обработки» справа. Реализовать интерфейс, позволяющий по нажатию на соответствующие кнопки выполнять следующие операции:

1. Предварительная обработка изображения для устранения шумов и перепадов яркости.
2. Поиск и отображение аппроксимированных контуров на изображении.
3. Поиск и отображение примитивов на изображении.
4. Возможность регулирования пользователем параметров порогового преобразования и минимальной площади контуров.
5. Отображение числа примитивов на изображении выбранного вида.